Adaptive Control and Reinforcement Learning (16-899C, S09) Lecture #3 (22-Jan-09)

Linear Programming, Dynamic Programming and Value-Iteration

Lecturer: Drew Bagnell *Scribe: Brian C. Becker*

1 Hwk#1: Tetris

- The scoring function is lines removed/rows cleared and you do not get a bonus for clearing multiple rows at a time.
- The AI can take as long as it wants to place the piece (in another words, the game does not speed up it progresses).
- There is no slide over. When a piece touches another piece, it freezes in place.
- The AI can rotate the piece above the board so you don't run into problems with insufficient space to rotate.
- Use the standard height of 20 and width of 10.
- A good AI should be able to clear about 100,000 rows.
- You can use any learning algorithm you want; in fact you can hand code heuristics.
- The game should only implement a one piece placing. The AI should not know what the next piece is going to be.

2 LQR: Finite Horizon

The general form for a Linear Quadratic Regulator (LQR) with a finite horizon is

$$u_t^T = -x_t^T K_t \tag{1}$$

$$K_t = A^T V_{t+1} B_t (R_t + B_t^T V_{t+1} B_t)^{-1} \tag{2}$$

$$V_t = Q_t + K_t^T R_t K_t + (A_t + B_t K_t)^T V_{t+1} (A_t + B_t K_t) \tag{3}$$

Where u_t is the input to the system for the current time step t, x_t is the current state of the system, K_t is the Kalman Gain, and V_t is the cost of being in the current state. Q_t and R_t are positive definite or positive semi-definite costs on the state and input, respectively. A_t and B_t describe the system dynamics and form the state-space model $x_{t+1} = A_t x_t + B_t u_t$ in discrete time. Often, A_t, B_t, Q_t, and R_t don't change with time (the costs and system dynamics don't vary from instant to instant) and can be denoted as A, B, Q, and R.

The problem of LQR can be defined as follows: given a set of costs on the state (Q) and the input (R) and a final desired state x_T, an optimal sequence of inputs $U \in \{u_0, \ldots, u_T\}$ can be found by starting with the input for the final state and working backwards to the current state. See previous lecture for more details.

2.1 Advantages

The advantage of LQR is that it finds optimal solutions as defined by your costs Q and R. It also handles Multi-Input Multi-Output (MIMO) systems much better than say a PID controller. MIMO systems are often highly coupled, leading to a nightmarish back and forth tweaking of PID values for the various subsystems. LQR handles all this for you in the costs and state-space dynamics.

2.2 Disadvantages

There are several instances where the standard finite horizon LQR formulation fails, including systems involving stochasticity, infinite horizons, nonlinearities in system dynamics, non-quadratic cost functions, and missing states. Each will be briefly examined.

2.2.1 Stochasticity

In a stochastic system, we expect a certain amount of noise or uncertainty in the system. Noisy sensors incorrectly measure positions, velocities, and forces while noisy actuators cannot move the robot to the commanded position or velocity with perfect accuracy. To cope with this uncertainty, we add a Gaussian noise term to our model:

$$x_{t+1} = A_t x_t + B_t u_t + \Delta \tag{4}$$

$$y = C_t x_t + \delta \tag{5}$$

where $\Delta \sim N(0, \Sigma)$ and $\delta \sim N(0, L)$. We'll discuss stochastic observations more when we get to the topic of POMDPs.

2.2.2 Nonlinear System Dynamics

If the system is inherently nonlinear, you can simply linearize the system about a trajectory and use LQR. In particular, linearizing using a first order Taylor series expansion is useful for trajectory following. If this is insufficient, the fix is to use Differential Dynamic Programming, which will be covered later.

2.2.3 Non-quadratic Cost Function

Sometimes the cost functions are non-quadratic, especially on real-world systems. LQR assumes that the system can actuate any input where in reality actuators have limits. For instance, LQR

may find an optimal sequence of inputs for the system, but some inputs exceed the torque limits of the motors. The solution for this is usually to place a cap on the inputs LQR finds or adjust the Q and R matrices so this does not happen.

2.2.4 Missing states

Frequently, sensors will not give you all the states you need. You might have a noisy position sensor and no velocity sensor. This is unfortunate because your state depends on the velocity as well as the position. In this case, you can try to estimate the missing state through numerical differentiation. The results are typically unsatisfactory because of noise. A smoothing lowpass filter or a Kalman filter can help estimate the states with higher accuracy.

2.2.5 Infinite Horizon

In the case of pathplanning, the final state is a static configuration at a static location. However in other situations, there may be no final static state you want to reach. For instance, in the case of an inverted pendulum, you want to keep running the control forever. As $T \to \infty$, it becomes computationally infeasible to evaluate the optimal control inputs. The problem here is that for controlling an inverted pendulum under noise, $\lim_{T\to\infty} V_t$ will go to infinity.

The first option is to simply keep iterating. Take T to be a very large number and start working backwards. For this to work, the following limit must exist:

$$\lim_{T\to\infty} V_t \tag{6}$$

In standard pathplanning, $V_t \to 0$ as the robot approaches and finally reaches the goal point. In order for this to work, we need a final absorbing state that uses zero cost to remain in the state. Even if the above limit exists and is finite, there are cases where this approach does not work. For instance, if V_t has a finite limit, but cycles between states.

The formulation of the general infinite horizon MDP is:

$$V^*_\infty(s) = r(s) + \min_a \mathbb{E}_{P(s'|s,a)}[V^*_\infty(s')] \tag{7}$$

where $V^* = x^T V x$. The value can be represented by a quadratic as in $x^T V x$ for LQR. Then you can get the stationary optimal policy π^*_∞:

$$\pi^*_\infty = \arg\min_a [r(s,a) + \mathbb{E}_{P(s'|s,a)}[V^*_\infty(s')] \tag{8}$$

2.2.6 Receeding Horizon

The second option for solving the infinite horizon case is to use a receding horizon. First, plan 5 steps ahead, take one step, then plan another 5 steps ahead, etc. By repeatedly using a small horizon, you can ensure that your control inputs are at least optimal for some distance into the

future. This is particularly useful when it is costly to compute more than several time steps into the future and you still need realtime control over the robot.

2.2.7 Introducing a Final Absorbing State

A third option is to add a discount factor to ensure that $\lim_{T\to\infty} V_t$ exists. In essence, you are artificially enforcing a final absorbing state associated with a cost of zero even if it doesn't exist. In the inverted pendulum example, you never have a final state where the cost is zero. You will always have to have some input to counteract gravity and other disturbances such as wind. However, by artificially introducing a "dead" state where no more control input is needed (reward is zero), we can solve the LQR formulation. To make this work, we add a transition probability $\gamma \in [0, 1]$ that specifies the chance of NOT transitioning to the dead state. Once in the dead state, the system cannot get out. This is a handy trick to make sure there is a final absorbing state. The formulation then becomes:

$$V_\infty^*(s) = r(s) + \gamma \min_a \mathbb{E}_{P(s'|s,a)}[V_\infty^*(s')] + (1-\gamma)V_{DEAD} \tag{9}$$

Because the reward of the final absorbing state is zero, $V_{DEAD} = 0$ and the final term can be neglected. By solving this equation iteratively using value iteration, we can obtain the optimal policy.

2.2.8 Policy Iteration

Another more effective way to solve this is to use policy iteration where you select an initial policy π_0 to execute at $T-1$ and iterate:

$$V_{T-2}^{\pi_0} = r(s) + \gamma \mathbb{E}[V_\infty^{\pi_0}] \tag{10}$$

This formula is linear in V and can be solved with:

$$(I - \gamma T_{\pi_0}) V_\infty^{\pi_0} = r(s) P(s'|s, \pi_0(s))$$

$$\pi_1(s) = \arg\max_a \mathbb{E}_{P(s'|s,a)}[V^{\pi_0}(s')] \tag{11}$$

where T_{π_0} is the time when executing the last policy before steady state. This converges to a globally optimal solution in a finite number of iterations because there are a finite number of policies. This is guaranteed to do as well as value iteration at each iteration. Realistically, nobody does value iteration anymore for discrete MDPS because policy iteration is so much better. In a modified version of policy iteration, you can start with the last answer. One thing to keep in mind is while V is unique, the policy generated might not be unique. That is to say, if trying to plan a path around a rock, it might be equally good or bad to go around the rock to the left or to the right.